IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

TITLE: METHOD AND APPARATUS FOR STORAGE TANK LEAK DETECTION

INVENTORS: JIMMY WOLFORD, BERNIE WOLFORD, CLARK LOCKERD, RICKY SLAUGHTER

BACKGROUND OF THE INVENTION

1. Field of the Invention.

The present invention is directed towards a method and apparatus for providing a safe, precise, and cost-effective storage tank leak detection system; and more particularly, to a method and apparatus wherein the containment integrity of a storage tank is determined by mass measurements of the stored product.

2. Background Information.

Storage tanks play a vital role in today's economy. The economy, on a global scale, depends on the proper function of these tanks as they are prevalent in several industries and virtually every geographical region in the world. In light of the vital role these storage tanks play, the integrity of the tanks is placed at a premium. That is, storage tank owners are willing to invest huge sums of money in both the maintenance and inspection of such tanks.

These tanks come in all shapes and sizes, are found both below and above ground, and are used to store a wide-range of materials. Storage tank capacities range from hundreds to millions of gallons and are used to store a staggering assortment of products; these storage tanks are commonly used to store hazardous material.

As one could imagine, there is a wide range of problems associated with maintaining storage tank integrity, particularly with above ground storage tanks. Given the enormous dimensions of above ground tanks, the corrosive products contained within the tanks, the incredible mass of the stored product, and the extreme weather conditions the tanks are subjected to; it is plain to see that above ground storage tank leaks are an all-to-common problem. Using the United States Environmental Protection Agency leak detection threshold criteria of .05 gallons per hour in a 10,000-gallon underground tank, that threshold would equate to a 15 gallon per hour detection level in an 80,000 barrel above ground tank. Given the limited number of systems capable of meeting the EPA's underground storage tank leak detection threshold and the added difficulties associated with above ground tanks, the difficulty in protecting against and detecting leaks is easily seen.

However, the recognized difficulty in preventing storage tank leaks does not mitigate the duties or liabilities imposed on responsible parties. Tremendous environmental and economic consequences and the threat of litigation and clean up costs associated with storage tank leaks force responsible parties to invest large sums of money in the maintenance and

inspection of the tanks. Tank inspections are costly with respect to the amount of money spent, the danger presented to the inspectors and the environment, and production downtime. In fact, these inspections often remove a tank from service for more than one month. The threat of liability also forces responsible parties to spend money unnecessarily for the maintenance of these tanks. Moreover, liability does not end with litigation and clean-up costs.

Currently, responsible parties are, in some countries, being incarcerated as a direct result of storage tanks leaks. These leaks have contaminated surrounding ground water, some of which serves as drinking water for local residents. As such, the facilities associated with such incidents have been shutdown until compliance with emissions regulations can be established beyond reasonable doubt. Such proof, in turn, is dependent on proof of reliable and sufficiently accurate detection systems and methods for proving such compliance. Each day the shuttered facilities remain inoperative adds to an already tremendous amount of money lost.

Prior to the present invention (to be described in detail hereafter), there are simply no known systems or methods by which the leak detection requirements can be met. Presently available leak detection systems lack detection thresholds low

enough to detect leaks down to permissible upper leakage limits for above ground storage tanks.

Clearly, for the reasons set forth above, there is a dire and immediate need for the ability to determine, with far more precision than presently possible through use of presently available systems and methods, the presence and degree of leakage from above ground storage tanks, at least to the extent of proving compliance with applicable storage tank leakage regulations or statutes.

Comparison with Known Technologies in the Field

Storage tank leak detection systems are known in the art; however, these products are fraught with problems. The present systems are imprecise, or provide erroneous data for any or all of reasons including: the consistency of the soil acting as the tank's foundation, the temperature stratification of the in-tank product, extraneous noise sources, thermal expansion of the tank's contents, water table level, previous soil contamination, and/or tank shell dynamics.

Further, some detection devices can only be used when the storage tank is empty, and no known system or method ensures a comprehensive inspection of the tank. The most common form of such a system is "vacuum box testing;"

however, this system is intended only for weld joints and is not usually applied to the entire tank bottom. Magnetic flux floor scanning is also used, but is not effective at examining the area of the floor surface close to the surface walls or where there are physical obstructions. Ultrasonic detection is used, but this is only effective for small areas of the surface. Gas detection is also used, but the types of materials stored in the tank can obstruct this method.

Other common leak detection systems employ a level sensor. However, even large volume changes produce only small level changes, as the cross-sectional area of the liquid surface in these tanks is very large. This, combined with differential expansion and temperature change of the stored liquid and its vapor, make this type of detection system inconsistent and very nearly worthless.

Finally, mass measurement detection systems are known in the art. However, the presently available systems and associated methods are not capable of the precision, which is indicated above as crucial at the present time (and which, as described below, is afforded by the systems and methods of the present invention). Present mass measurement leak detection systems in the art are limited by tank shell

variations resulting from temperature effects on tank shell plating. As such, known mass measurement detection systems are only sensitive enough to be used in smaller tanks, typically underground storage tanks. However, as will be seen in the specification to follow, the present invention overcomes tank shell variations and other shortcomings of presently known technology in this field through data collection and data correction apparatus, techniques and interpretation.

In light of the severe consequences of failing to detect significant storage tank leaks, presently not detectable through use of known systems or methods, there is a compelling need for a system and method by which one can detect very small leaks even in very large tanks, ideally in a safe and cost effective manner.

It would well serve those who are responsible for maintaining storage tank integrity to provide a safe, precise, and cost-effective detection system that does not depend on independent variables such as fluid temperature, fluid stratification, or tank stabilization, and may be used in an efficient manner thereby preserving industrial and environmental resources.

SUMMARY OF INVENTION

In view of the foregoing, it is an object of the present invention to provide a storage tank leak detection apparatus with a very low detection threshold that may be used in an efficient manner thereby preserving industrial and environmental resources.

It is another object of the present invention to provide an apparatus for safe storage tank leak detection

It is another object of the present invention to provide an apparatus for precise storage tank leak detection

It is another object of the present invention to provide an apparatus for cost-effective storage tank leak detection

It is another object of the present invention to provide an apparatus for non-intrusive storage tank leak detection

It is another object of the present invention to provide an apparatus for storage tank leak detection where the contents of the storage tank do not have to be removed

It is another object of the present invention to provide an apparatus for storage tank leak detection where no chemical additives are involved

It is another object of the present invention to provide an apparatus for immediate storage tank leak detection

It is another object of the present invention to provide an apparatus for conclusive storage tank leak detection

It is another object of the present invention to provide an apparatus for quantitative storage tank leak detection

It is another object of the present invention to provide an apparatus for storage tank leak detection that does not depend on fluid temperature changes

It is another object of the present invention to provide an apparatus for storage tank leak detection that does not depend on fluid stratification

It is another object of the present invention to provide an apparatus for storage tank leak detection that does not require tank stabilization time

It is another object of the present invention to provide an apparatus for storage tank leak detection that requires only minimal tank preparation

It is another object of the present invention to provide an apparatus for storage tank leak detection that

has been evaluated by an EPA-recognized, independent third party laboratory

It is another object of the present invention to provide a method with a very low detection threshold that may be used in an efficient manner thereby preserving industrial and environmental resources.

It is another object of the present invention to provide a method for safe storage tank leak detection

It is another object of the present invention to provide a method for precise storage tank leak detection

It is another object of the present invention to provide a method for cost-effective storage tank leak detection

It is another object of the present invention to provide a method for non-intrusive storage tank leak detection

It is another object of the present invention to provide an apparatus for storage tank leak detection where the contents of the storage tank do not have to be removed

It is another object of the present invention to provide an apparatus for storage tank leak detection where no chemical additives are involved

It is another object of the present invention to provide a method for immediate storage tank leak detection

It is another object of the present invention to provide a method for conclusive storage tank leak detection

It is another object of the present invention to provide a method for quantitative storage tank leak detection

It is another object of the present invention to provide a method for storage tank leak detection that does not depend on fluid temperature changes

It is another object of the present invention to provide a method for storage tank leak detection that does not depend on fluid stratification

It is another object of the present invention to provide a method for storage tank leak detection that does not require tank stabilization time

It is another object of the present invention to provide a method for storage tank leak detection that requires only minimal tank preparation

It is yet another object of the present invention to provide a method for storage tank leak detection that has been evaluated by an EPA-recognized, independent third party laboratory.

The present invention provides a safe, extremely precise, and cost-effective solution to the problems mentioned above. Test results associated with the present invention provide an accurate determination of containment integrity, and in the event of leakage, a precise volumetric leak rate. The present invention is not restricted by fluid type, fluid temperature, fluid level, or tank size.

Distinguished from products known in the art, the present invention provides an intrinsically safe detection system. The leak detection system of the present invention uses a sufficiently low wattage (as established in the National Electric Code) so that the components of the system may be placed within the class I area of the tank. In fact, the present invention provides for leak detection system components to be placed within the storage tank. As will be described in the specification to follow, placement of leak detection components in the tank used in combination with system control techniques and data correction software, provide for precision not possible with products known in the art.

Further, no physical inspection of the tanks is required for practice of the present system. As such, there is no need to drain, clean, or enter the tank. With no need

for physical inspection, neither inspectors nor the environment are exposed to the contents of the tank. With no need to drain the storage tank, practice of the present invention does not produce hazardous by-products associated with the draining/cleaning process, and danger from transport and storage of the drained product is avoided. Finally, the systems and methods of the present invention do not require chemical additives to be mixed with the tank contents. As such, incidental spills and leaks are avoided altogether.

Practice of the present invention is cost effective. Tank structure or the foundation and surrounding soil are not disturbed, as such; set-up time and capital investment costs are minimized. The present invention is non-intrusive and does not require manual inspection of the tank. Therefore, operation of the tank is not hindered, so there is no production downtime. There is no cost related to the handling, transport, disposal, or storage of removed hazardous material. Finally, testing can be accomplished simultaneously to further reduce the total time involved and rapidly identify problem areas.

The determinative feature of mass measurement leak detection systems is the sensitivity of the apparatus. That

is, the lower the leak detection threshold level of a device, the more effective it will be at detecting leaks. The present invention, by employing a combination of techniques and components not known in the art, provides a leak detection threshold that is much lower than any known device. Most importantly, the system of the present invention provides for placing mass measuring components within the actual storage tank, thereby eliminating extraneous noise associated with bubbler units required by other products in the art. The system secures the mass measuring components within a vacuum and holds the mass measurement component's temperature constant during the entire measurement process. Further, the system corrects errors in the data attributed to storage tank shell dynamics and inherent imprecision in the mass measurement devices. This data correction process will be discussed in detail in the specification to follow.

As mentioned, tank shell variations limit the effectiveness of presently known mass measurement detection systems. The systems and methods of the present invention overcome tank shell variations through data collection and data correction techniques. First, data is collected through use of a quartz crystal type pressure transducer

(the specifications and use of this transducer will be explained in more detail in the Detailed Description of the Preferred Embodiment). An intrinsically safe remote terminal unit, connected to the pressure transducer, records pressure data over a period of time (preferably one to five nights). The atmospheric temperature and barometric pressure are recorded and precisely analyzed to calculate any changes in the mass of the fluid within the tank. This data is regressed to give a line slope that is converted to a leak rate, usually in gallons per hour.

Data generated by the transducer is collected on a 24-hour basis. Only data containing a sufficiently low amount of extraneous noise is analyzed. Such data is usually obtained at nighttime and during fair weather conditions. Also, data correction software accounts for the coefficient of expansion for any given storage tank. This correction eliminates the effect of the sun's radiant energy on the area of the surface tank, which may adversely affect the mass measurement of the stored product. The nighttime data is corrected for atmospheric conditions and variations in the tank shell. These measurements and corrections allow the system to repeatedly achieve the stated accuracy in real world conditions on a routine basis.

For even greater precision, the leak detection system of the present invention provides for an independent barometric measuring means to constantly record the barometric pressure during the data collection process. This independent barometric pressure measuring means used in combination with data correction software, corrects any zero drift associated with the individual pressure transducer. That is, this system corrects for the inherent error present in any transducer when that transducer deviates from its initial calibration.

Practice of the apparatus involves securing a combination of precise mass measurement components, including a highly precise quartz crystal type pressure transducer, in a vacuum-sealed canister. This canister is then lowered to the bottom surface of a storage tank. A differential reference is placed just above a liquid surface. The pressure, measured at the tank floor ("tank bottom pressure") and atmospheric pressure measured just above the liquid surface, is recorded by the above-referenced micro sensitive differential pressure transducer, recorded on a real time basis and post processed using a data analysis routine to accurately calculate any changes in the mass of fluid contained within the tank to determine if

there is a loss. The present system, using the specified transducer, and when used in the manner and with the data interpretation described herein, is capable of detecting above ground storage tank leaks at a threshold of less than .9 gallons per hour with a probability of detection of 95% in a 100,000 barrel tank - far more accurate than possible with any presently available quantitative leak detection system. This, quantitatively, amounts to detecting pressure differentials equivalent to less than 1/10,000th inch of water column pressure, a tolerance level necessary to achieve such detection thresholds.

The method and apparatus of the present invention provides a safe and effective way to detect very small leaks in very large tanks. Particularly, the present invention provides a tremendous improvement in accuracy and leak detection threshold, allowing its users to achieve greater results than presently thought possible.

Thus, in satisfaction of the above objects, an embodiment of the present invention provides systems and methods for solving each of the stated problems with presently available storage tank leak detection systems.

BRIEF DESCRIPTION OF THE DRAWINGS

Annex A is a printout of the computer program source code referred to herein as the RTU program.

Fig. 1 is a block diagram depicting the general layout of the present leak detection system.

Fig. 2 is an elevational, sagital cross sectional view of the canister ("protective enclosure means") of the leak detection system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In the drawings and the description that follows, referring to figure 1, a preferred embodiment of a storage tank leak detection system according to the present invention is generally designated as system 10. An embodiment of the present invention is shown to include a vacuum-sealed canister 12, which houses and protects a plurality of mass measurement components and system control components. In the preferred embodiment, vacuum-sealed canister 12 is made of a substantially non-corrosive metal (aluminum, for example), however, any material that is corrosion resistant and offers sufficient protection to the components enclosed is adequate for use with the present invention. Canister 12 is directly immersed in storage tank 60 and rests on storage tank bottom surface 62. Canister 12 further contains vacuum seal nozzle

14 and transducer high side aperture 20. Vacuum seal nozzle 14 allows communication means to pass from the inside of the canister to the outside of the canister while maintaining the integrity of the vacuum inside the canister. Vacuum nozzle 14 further contains barometric pressure measuring means aperture 16 and transducer low side aperture 18.

At its proximate end canister hose 15 forms a fluid tight seal with vacuum seal nozzle 14. Extending from vacuum seal nozzle 14, canister hose 15 passes through storage tank top surface recess 64 to an area outside of the class I region of storage tank 60 (class I region refers to the National Electric Code designated hazardous areas in which only power wattage levels of less than certain prescribed levels may be introduced). Canister hose 15 serves as a conduit for communication means extending though vacuum nozzle 14 and as an atmospheric reference in its service as a barometric pressure measuring means reference hose. Canister hose 15 allows transducer low side aperture 18 and barometric pressure measuring means aperture 16 to be directly exposed to atmospheric pressure while maintaining a fluid tight seal with vacuum seal nozzle 14 thereby preserving the integrity of the vacuum of canister 12.

Contained within vacuum-sealed canister 12 is differential pressure transmitter 22. In the preferred embodiment, differential pressure transmitter 22 is comprised of a highly precise quartz crystal type pressure transducer 24. Transducer 24 contains an oscillating quartz crystal and has a resolution of 1x10-8, as known in the industry. Such a unit is available from Paroscientific, Inc. as Model No. 6015-G. The ultimate resolution achievable with a transducer is limited by its noise level. System 10 greatly reduces noise thereby increasing the resolution of transmitter 24. In system 10, transmitter 22 has been modified from its original configuration so that it may be directly immersed in storage tank 60. This modification has eliminated dependence on any bubbler unit (thereby eliminating noise associated with such units) as required by other products. As will be further described in this section, transducer 24 is held at a constant temperature and secured in vacuum to further reduce noise.

Quartz crystal type pressure transducer 24 is further comprised of transducer low side 26. Transducer low side 26 is a differential reference that receives the barometric pressure value at the liquid surface. Transducer low side tube 28 forms an air tight seal at it proximate end with transducer low side 26 and extends though the vacuum of

canister 12 where it forms an air tight seal at its distal end at transducer low side aperture 18 of vacuum seal nozzle 14. Transducer low side tube 28 allows transducer low side 26 to receive the barometric pressure from the reference point at the liquid surface while allowing canister 12 to remain in vacuum.

Quartz crystal type pressure transducer 24 is further comprised of transducer high side 30. Quartz crystal type pressure transducer high side 30 is a pressure reference point, which measures the sum of the barometric and hydrostatic pressure at tank bottom surface 62. Transducer high side 30 contains a protruding transducer high side tube 32. In the preferred embodiment, transducer high side tube 32 is filled with a pressure-sensing liquid and extends through transducer high side aperture 20 where it is ported to the product contained in tank 60. Transducer high side tube 32 is surrounded by tube fitting 34. In the preferred embodiment, tube fitting 34 slides along high side tube 32 and forms a fluid tight seal at high side aperture 20. Tube fitting 34 allows high side tube 32 to extend through high side aperture 20 while maintaining the integrity of the vacuum of canister 12.

Transducer 24 subtracts the value received at transducer low side 26 from the value received at transducer high side 30 to arrive at the pressure exerted by the mass of the stored product. Transmitter 22, communicating digitally, then sends this processed information to data-logging computer 30. This data is transmitted along data transfer means 23. In the preferred embodiment, data transfer means 23 is a standard bus communications cable. However, one could easily envision a data transfer means such as wireless communication that would work equally as well. Data transfer means 23 extends from the output of differential pressure transmitter 22 through vacuum seal nozzle 14 and continues, separated from storage tank's 60 contents by canister hose 15, to data logging computer 30.

Also contained within canister 12 is current transmitter 34. Current transmitter 34 serves as a part of a temperature regulation scheme used to keep the contents of canister 12 at a constant temperature during the data gathering process. Current transmitter 34, in the preferred embodiment, actuates a resistive heater 36 by a simple on/off control loop. Heat sink 38, acting in combination with current transmitter 34 and resistive heater 36 acts to regulate the temperature of canister 12. While the above temperature regulating scheme has been described with reference to one embodiment, one could

easily imagine other temperature regulation schemes that would work equally as well. Data transfer means 39 extends from the output of current transmitter 34 through vacuum seal nozzle 14 and continues, separated from storage tank's 60 contents by canister hose 15, to data logging computer 30. In the preferred embodiment, data transfer means 39 is a standard bus communications cable. However, one could easily envision a data transfer means such as wireless communication that would work equally as well.

The use of this temperature regulation scheme to hold transmitter 22 at a constant temperature further increases the precision of the current apparatus. The absolute temperature at which transmitter 22 is maintained is not critical, rather constancy of temperature affects the integrity of the subject measurements. As a matter of practicality and economy, temperature of transmitter 22 is maintained, according to the presently preferred mode of the present invention, at a temperature of approximately 1° F above the ambient temperature of the product (oil or gasoline, for example) in tank 60. If, for example, the product is at 50° F, transmitter 22 is maintained at 51° F, if the product is at 90° F, transmitter 22 is maintained at 91° F, and so forth.

Also contained within canister 12 is barometric pressure measuring means 40. Barometric measuring means 40 serves as an independent reference for true atmospheric pressure. In the preferred embodiment, barometric pressure measuring means 40 may be any standard barometer that sends signals to be processed by data logging computer 30. Barometric measuring means 40 is very useful for increasing the precision of system 10. All transducers decrease in accuracy over time as they lose their calibration with respect to true atmospheric pressure. This is known as zero drift. However, the present invention employs barometric measuring means 40 to serve as an independent measure of true atmospheric pressure thereby allowing for data correction over any extended period of time. As will be discussed in this section, data correction using values taken from barometric pressure measuring means 40 is software based and greatly increases the precision of the current invention.

Barometric measuring means tube 42 forms an air tight seal at it proximate end with Barometric measuring means 40 and extends though the vacuum of canister 12 where it forms an air tight seal at its distal end at barometric measuring means aperture 16 of vacuum seal nozzle 14. Barometric measuring means tube 42 allows barometric measuring means 40 to receive

the barometric pressure from the reference point at the surface of liquid within storage tank 60, while allowing the interior of canister 12 (with transmitter 22 installed therein) to remain in vacuum so as to substantially eliminate any environmentally-effected variations in instrument performance). Data transfer means 43 extends from the output of barometric pressure measuring means 40 through vacuum seal nozzle 14 and continues, separated from storage tank's 60 contents by canister hose 15, to data logging computer 30. In the preferred embodiment, data transfer means 43 is a standard bus communications cable. However, one could easily envision a data transfer means such as wireless communication that would work equally as well.

Although not necessary, remote computer 30 is typically housed in a separate enclosure, such as field unit 50. In accordance with the described routines to follow and the exemplary computer code depicted in Annex A attached hereto and incorporated herein by reference, data logging computer processes data received from transmitter 22, current transmitter 34, resistive heater 36, heat sink 38, and barometric pressure measuring means 40. Data logging computer 30 communicates with remote computer 70 by data transfer means 72.

The software commences operation with the initialization of data collection at the tank bottom, along with the atmospheric and environmental conditions. Data is automatically collected via computer controlled programming over some length of time, preferably 36 to 60 hours. The length of the test is dependent on tank size and site atmospheric conditions. In the preferred embodiment, data transfer means 72 is a standard bus communications cable. However, one could easily envision a data transfer means such as wireless communication that would work equally as well.

As will be discussed and illustrated hereafter, remote computer 70 contains software that performs linear regressions of data received from data logging computer 30. This regression detects minuscule changes in the mass of the stored product, thereby indicating the presence of the smallest of leaks. As the compilation of data grows, the more precise the regression becomes. The post processing module and software of remote computer 70 is independent of the data-logging computer 30.

There are two software programs or modules involved with the storage tank leak detection system of the present invention: The RTU program and the linear regression program.

The RTU program is performed by data logging computer 30

and is responsible for obtaining (routine 100) and correcting (routine 200) pressure readings from transmitter 22, controlling the temperature of transmitter 22 (routine 300), calculating adjustments for tank shell expansions (routine 400), obtaining transmitter 22 temperature (routine 500), and data storage. The data acquired by the RTU program is stored within data-logging computer 30 in non-volatile memory 31.

The purpose of the RTU program is to interrogate an intelligent differential pressure transmitter (transmitter 22) via a serial connection. The pressure read from transmitter 22 is the difference in pressure read from transducer low side 26 and transducer high side 30. That pressure value is modified by two additional variables in order to improve the accuracy of the reading. The program performs correction of barometric pressure; an analog barometer (such as barometric pressure measuring means 40) provides the signal that is sent to correct transmitter pressure for errors due to changes in barometric pressure, as measured at the upper surface of the contents of storage tank 60. Also, the program monitors ambient temperature to compensate for changes in the tank diameter which otherwise would skew the data interpretation, intended solely to detect variations of contents of storage tank 60 due to leakage. Any change in tank diameter is

accommodated in the calculations of transmitter 22, thus properly attributing substantially all variations in differential pressure (already corrected for variations in atmospheric pressure, as mentioned above) to variations in the content of storage tank 60, such as through leakage.

Routine 100, obtaining pressure readings from transmitter 22, is performed every one minute as follows: at step 101 a command is sent to transmitter 22 to obtain a new pressure sample, at step 102 a command to wait for the new sample is sent, at step 103 a command to ask for the new sample is sent. The data is returned as an ASCII psi number.

Routine 200, adjusting transducer 24 pressure reading according to barometric measuring means 40, is performed as follows: At step 201 the user enters installation parameters reflecting: (1) a predetermined barometric correction factor which is laboratory-determined for each transmitter 22 to establish, and to later enable calibration of the transmitter 22's "zero point"; (2) the coefficient of expansion for storage tank 60 (a factor readily calculated by persons reasonably skilled in the relevant field, applying common materials engineering principles to standards pertaining to storage tank design and materials); (3) the ambient temperature at the installation site at the time installation

(for use in calculating dimensional variations in tank 60 according to the aforementioned coefficient of expansion; and (4) the specific gravity of the product contained in storage tank 60. The user will also enter the desired temperature setting for resistive heater 36.

At step 202 the following calculations are undertaken:

$$Hadj = H - Bcf * (B - 14.5)$$

$$Hadj / sg$$

where (for the present discussion, although not precisely reflected in the same terms in the appended source code) Hadj = adjusted or derived head pressure; H = head pressure in water feet (Hpsi x 2.037); Hpsi = head pressure in pounds per square inch (measured at high side 30 of transmitter 22); B = barometer reading in pounds per square inch; sg = specific gravity of the content of tank 60; and Bcf = the barometric correction factor. The number 14.5 is a somewhat arbitrary number which is fairly close to an expected range of actual, measured barometric pressure. This factor is subtracted from measured barometric pressure in order to reduce certain calculated figures to a smaller, and more manageable level for later processing (linear regression, etc.) in tracking minute mass differences in storage tank contents. At step 203 this

adjusted or derived pressure data is used to calculate the mass of the product in the tank based on the tank's diameter.

Routine 300, controlling transducer 24 temperature, is performed as follows: at step 301 the digital output to current transmitter 34 is turned on when the temperature read from analog input of heat sink 38 is .1 degree below the temperature set point, at step 302 the digital output to current transmitter 34 is turned off when the temperature is 0.1 degree above the set point.

Routine 400, adjusting the previously derived content mass for tank shell expansion, is performed as follows: at step 401 the ambient temperature is averaged to obtain a temperature to use in calculating the change in tank diameter, this calculation requires the coefficient of expansion and the tank diameter to be entered by the user either at startup (as mentioned previously) or at any time, the result obtained is used to adjust the total mass in the tank for erroneous, environmentally effected false indications of changes in the content of tank 60, to yield purely leakage related variations (assuming no intension addition or removal of contents by other means).

Routine 500, obtaining transducer 24 temperature, is performed as follows: at step 501 a command is sent to

transmitter 22 to obtain a new temperature, at step 502 a command to wait for the temperature reading is sent, at step 503 a command to ask transmitter 22 for the new reading is sent.

Finally, the RTU program is responsible for data storage. The amount of data storage available will determine how many days of data are stored for retrieval. One record per minute is stored. The organization of the date is by days. The record for every minute will include: (1) the tank contents in pounds (as a floating-point number, IEEE 32 bit format), (2) the barometric pressure (as a x100-16 bit integer), (3) the ambient temperate (as x100-16 bit integer). Other data, such as previous transducer temperatures, tank diameter, and tank coefficient of expansion, may also be stored as current data. The second software program of the storage tank leak detection system of the claimed invention is the linear regression program. Remote computer 70 performs this program. Routine 700, linear regression of received data, is performed as follows: at step 701 the data file created by the RTU program is created and the leak analysis is performed, at step 702 the data sections are selected for the quality of weather during that particular data section- only nighttime data are typically used in order to minimize extraneous noise in the

analysis, at step 703 a best linear fit is used for data points in each data section- when the sections of data that represent durations of appropriately low noise level are included in the best fit data regression, the slope of the best fit line indicates the leak rate. Calculation of the linear regression and best fit are straightforward and could be performed by common software such as Microsoft Excel.

It is believed that, while safe and efficient, the present device will obviate significant inconvenience and provide substantial utility to those who wish to detect leaks in storage tanks. Specifically, the present device will allow very small leaks to be detected in very large storage tanks in a consistent and cost-effective manner.

Although the invention has been described with reference to specific embodiments, this description is not meant to be construed in a limited sense. Various modifications of the disclosed embodiments, as well as alternative embodiments of the inventions will become apparent to person skilled in the art upon the reference to the description of the invention. It is therefore contemplated that the appended claims will cover such modification that fall within the scope of the invention.

ANNEX A

```
/*******************************************************************
*                                   *
* Mass Technologies - Data logger                   *
*                                   *
*
*******************************************************************/

// use 2700H for root memory reserve

#use eziobl17.lib
#use aasc.lib
#use aascz0.lib
#use aascz1.lib
#use aascscc.lib
#use vdriver.lib
#use msz.lib
#ue flashstore.lib

#define pi 3.1415927

/////////////////////////////////////////////////////////
//
// Configuration Data - stored in Flash
//
/////////////////////////////////////////////////////////


float calib[6][2] = {
//              {-.4979, 16.68},    // min,max for Baro sensor psi for 0-10V converter
                {0.0, 36.0},
                {00.0, 100.0},      //      for xducer temp degF (0-100)
                {00.0, 150.0},      //      for tanktemp 1 (0-150)
                {00.0, 150.0},      //      for tanktemp 2 (0-150)
                {00.0, 150.0},      //      for tanktemp 3 (0-150)
                {00.0, 150.0}       //      for tanktemp 4 (0-150)
               };

/////////////////////////////////////////////////////////
//
// Installation Data - stored in Flash
//
/////////////////////////////////////////////////////////
```

```
float desired_xducer_temp = 60.0;
float Bcf           = 0.004;
float coef_of_expn      = 1.000;
float tank_dia          = 60.0;
float sg            = 1.00;
float initial_temp              = 60.0;
float day_start         = 0000;

////////////////////////////////////////////////////////
//
// Logging Data - Current data stored in ram, previous days
//           stored in flash
//
////////////////////////////////////////////////////////

struct {                                    // holds 6 hours of logging data
  int initial1, initial2; // set to 1234, (rec#) when memory is initialized
  int year;
  int month;
  int day;
  int hour;
  int min;
  float mass[240];
  unsigned int baro[240];
  int avg_temp[240];
  int xducer_temp[240];
  } datastorage;

union { datastorage f; char flash[2560]; } data;

union { float f; unsigned int i[2]; } modbusd;

////////////////////////////////////////////////////////
//
// Runtime data storage -- it is not necessary to save
//          this between runs
//
////////////////////////////////////////////////////////

#define BUFFSIZE 64
struct _Channel *chan1;                   // channel for serial baro sensor
char readBuf1[BUFFSIZE];          // storage buffer for data from sensor
char writeBuf1[BUFFSIZE];         // storage for data to sensor
struct _Channel *chan4;                   // channel for serial to terminal
char readBuf4[BUFFSIZE];          // storage buffer for data from terminal
```

```
char writeBuf4[BUFFSIZE];            // storage for data to terminal
char eol[3] = {13, 10, 0};          // end of line (carriage return, line feed) string
char baro_cmnd[24];          // command to baro sensor
char baro_resp[64];                              // response from baro sensor
char serial_temp[24];                // latest temp from baro sensor
char serial_pressure[24];            // latest pressure from baro sensor
char term_in[64];           // line received from terminal
char term_out[64];                // data to send to terminal
char date_time[24];
shared float avg_tank_temp;  // average of the four tank temp sensors
shared float xducer_temp_min; //
shared float xducer_temp_max; //
shared float xducer_temp_avg; //
shared float xducer_ytemp_min;//
shared float xducer_ytemp_max;//
shared float xducer_ytemp_avg;//
int heater;
int start_hour, start_min;   //
int start_year;
char *flashptr;
int current_record;
int modbusaddr;

union { datastorage f; char flash[2560]; } old;

///////////////////////////////////////////////////////////
//
// Current sensor values -- updated as new data arrives
//
//
///////////////////////////////////////////////////////////

shared float cur_xducer_sensor;
shared float cur_xducer_temp;
shared float cur_local_baro;
shared float cur_pod_temp;
shared float cur_tank_temp_1;
shared float cur_tank_temp_2;
shared float cur_tank_temp_3;
shared float cur_tank_temp_4;

shared float mass;
shared float baro;

int year, month, day, hour, minute;
```

```
int monthdayhour;

///////////////////////////////////////////////////////////
//
// read date/time from RTC
//
//
///////////////////////////////////////////////////////////

void read_datetime(){
 struct tm time;

 tm_rd(&time);
 year = time.tm_year + 1900;
 month = time.tm_mon;
 day = time.tm_mday;
 hour = time.tm_hour;
 minute = time.tm_min;
 monthdayhour = month*1000 + day*100 + hour;
 return;
}

void write_datetime(){
 struct tm time;

 time.tm_year = year-1900;
 time.tm_mon = month;
 time.tm_mday= day;
 time.tm_hour= hour;
 time.tm_min = minute;
 tm_wr(&time);
}
///////////////////////////////////////////////////////////
//
// Read record# into root memory
//
//
///////////////////////////////////////////////////////////

void flash_read(int record){

  long a;

  a=flashstorage + (record*sizeof(data.flash));
```

```
  xmem2root(
         a,
         &old.flash,
         sizeof(data.flash)
         );
  return;
}

//////////////////////////////////////////////////////////
//
// Write current logging data into
//    flash memory at record#
//
//////////////////////////////////////////////////////////

void flash_write(int record){
  int r;

  r=WriteFlash(
           flashstorage+ ( record*sizeof(data.flash) ) ,
           data.flash,
           sizeof(data.flash)
       );
// if (r!=0)
//    printf("\nFlash error %d, writing record %d",r,record);
  return;
}

//////////////////////////////////////////////////////////
//
// Ready storage system for use
//
//
//////////////////////////////////////////////////////////

void flash_restart(){
  // go down flash storage, if any record is not yet intialized,
  // initialize it
  int i;

  for (i=0; i<36; i++) {
    flash_read(i);

    if (old.f.initial1!=1234 || old.f.initial2!=i) {
      old.f.year = 0;
```

```
    old.f.day = 0;
    old.f.hour = 0;
    old.f.initial1=1234;
    old.f.initial2=i;
    WriteFlash(
          flashstorage+ ( i*sizeof(old.flash) ) ,
          old.flash,
          sizeof(old.flash)
       );
   }
  }
  return;

}


//////////////////////////////////////////////////////////
//
// return index number for record
//
//
//////////////////////////////////////////////////////////

getindex(int h, int m){
 return ( ((h*60+m)%240) );
}


//////////////////////////////////////////////////////////
//
// Return index # for current data point
//
//
//////////////////////////////////////////////////////////

getcurrentindex(){

 read_datetime();
 return(getindex(hour,minute));

}

//////////////////////////////////////////////////////////
//
// Format a date/time string from
```

```
//   record # and index in a flash record
//  return: year-mm-dd hh:mm
//  if record # is < 0, use current (RAM) record
//  if record # is not <0, use flash record that should
//    have been copied to RAM as 'old'
/////////////////////////////////////////////////////

int formatdt(int recnum, int index) {
  int lyear, lmonth, lday, lhour, lminute;

  if (recnum < 0) {
    lyear = data.f.year;
    lmonth = data.f.month;
    lday = data.f.day;
    lhour = data.f.hour;
  }else{
    lyear = old.f.year;
    lmonth = old.f.month;
    lday = old.f.day;
    lhour = old.f.hour;
  }
  lminute = index;
  lhour = lhour + ((int)lminute/60);
  lminute = lminute % 60;
  sprintf(date_time, "%4d-%02d-%02d %02d:%02d",lyear,lmonth,lday,lhour,
              lminute);
  if (lyear<2001 || lyear>2100) lyear = 0;
  return(lyear);
}

/////////////////////////////////////////////////////
//
// Scale input values into engineering units
//
//
/////////////////////////////////////////////////////

float scale(float x, int sensor_number){
  float m, b;
  float result;

  if (sensor_number > 0) x = (x-2.0)*1.25; // base inputs>1 on 4-20mA sigs.

  m = (calib[sensor_number][1]-calib[sensor_number][0])/10;
  b = calib[sensor_number][0];
```

```
  result = m*x + b;

  //printf("scaled: %f\n", result);
  return result;
}

/////////////////////////////////////////////////////////
//
// Head correction for barometric pressure changes
//   returns corrected head in feet
//
/////////////////////////////////////////////////////////

float HcBaro
      ( float Hpsi,         // Head in psi
        float B,            // barometer in psi
        float sg                                 // specific gravity
      ) {

  float H;                 // head in feet

  H = Hpsi * 2.307;                                  // result in feet of water
  H = H - Bcf * ( B - 14.5 );    // result is in

  return H / sg;
}


/////////////////////////////////////////////////////////
//
// Head correction for temperature change
//   returns Head correction amount
//
/////////////////////////////////////////////////////////

float HcTemp
      ( float Di,          // initial diameter in feet
        float Ti,          // initial temp in deg F
        float T,           // current temp in deg F
        float sg,          // specific gravity
        float H,           // current Head in feet
        float fac          // coef of expansion
      )
 {
  float Ci;                // initial circ in feet
```

```
 float Cf;                  // final circ in feet
 float Df;                  // final dia in feet
 float Ai;                  // initial area ft^2
 float Af;                  // final area ft^2
 float dV;                  // change in tank volume

 Ci = Di * pi;
 Cf = Ci + ( Ci * 5.6e-6 * fac * (T - Ti));
 Df = Cf / pi;
 Ai = Di*Di / 4. * pi;
 Af = Df*Df / 4. * pi;
 dV = (Ai - Af) * H;
 return H - (dV / Af);
}

clear_local_store(){
 int i, h;

 read_datetime();
 h = hour - (hour % 4);

 data.f.initial1 = 1234;
 data.f.initial2 = -1;

 for (i=0; i<240; i++) {
  data.f.mass[i]=-1.0;
  data.f.baro[i]=0;
  data.f.avg_temp[i]=0;
 }
 data.f.year = year;
 data.f.month = month;
 data.f.day = day;
 data.f.hour = h;
 data.f.min = 0;
}


check_local_store(){
 int i, h;

 // check data store in ram -- if it is not correct for the current
 //  data, write to flash, and initial RAM

 read_datetime();
 h = hour - (hour % 4);
```

```
  if (  data.f.year != year || data.f.month != month
    || data.f.day!= day || data.f.hour != h
    || data.f.initial1 != 1234 || data.f.initial2 != -1)
  {
   flash_write(current_record);
   current_record = (current_record+1)%36;
   clear_local_store();
  }
}


main() {
  int pos;                          // used for baro serial input processing
  int i, index;
  int rr, ii;
  float f;
  union { float f; char c[4]; } u;
  float Hc;

  int lastminute;                                         // used to track time changes

  VdInit();
  eioBrdInit(0);


  if (current_record < 0) current_record = 0;
  current_record %= 36;

  // open channel for baro serial transmissions
  chan1 = aascOpen(DEV_Z1, 0, ASCI_PARAM_8N1+ASCI_PARAM_1200*8,NULL);
  aascSetReadBuf (chan1, readBuf1, sizeof(readBuf1));
  aascSetWriteBuf(chan1, writeBuf1, sizeof(writeBuf1));
  aascTxSwitch(chan1,1);
  aascRxSwitch(chan1,1);

  // open channel for serial to terminal
  chan4 = aascOpen(DEV_SCC, 0, SCC_B+SCC_8N1+SCC_1200*8,NULL);
  aascSetReadBuf (chan4, readBuf4, sizeof(readBuf4));
  aascSetWriteBuf(chan4, writeBuf4, sizeof(writeBuf4));
  aascTxSwitch(chan4,1);
  aascRxSwitch(chan4,1);

  // open up modbus comms
  msaZ0(modbusaddr,9600,0);              // modbus ascii Rs232, 9600,8,n,1
```

```
strcpy(term_out,"MTC data logger. v1.30 \r\n\r\n*");

read_datetime();
lastminute = minute;
xducer_temp_min = 9999.9;
xducer_temp_max = 0.0;
xducer_temp_avg = 0.0;
xducer_ytemp_min = 9999.9;
xducer_ytemp_max = 0.0;
xducer_ytemp_avg = 0.0;
cur_local_baro = 14.5;

start_hour = (int)(day_start/100);
start_min = (int)(day_start - (start_hour*100));

while(1==1) {

 // do analog input processing
 costate analog_inputs always_on {
  f =eioBrdAI(0);
  cur_local_baro = ((cur_local_baro*10)+scale(f, 0))/11.0;
  f =eioBrdAI(1);
  cur_pod_temp = scale(f,1);
  f =eioBrdAI(2);
  cur_tank_temp_1 = scale(f,2);
  f =eioBrdAI(3);
  cur_tank_temp_2 = scale(f,3);
  f =eioBrdAI(4);
  cur_tank_temp_3 = scale(f,4);
  f =eioBrdAI(5);
  cur_tank_temp_4 = scale(f,5);
       waitfor (DelayMs(1000));
 }

 costate data_processing always_on {
  read_datetime();
  if (lastminute != minute) {                // a new minute -- do processing

    // watch for new day -- reset avg/min/max values
    if (hour == 0 && minute == 0) {
     xducer_ytemp_min = xducer_temp_min;
     xducer_ytemp_max = xducer_temp_max;
     xducer_ytemp_avg = xducer_temp_avg;
     xducer_temp_min = 9999.9;
```

```
      xducer_temp_max = 0.0;
      xducer_temp_avg = cur_xducer_temp;
     }

     // record min/max/avg baro temp
     if (xducer_temp_min > cur_xducer_temp) xducer_temp_min = cur_xducer_temp;
     if (xducer_temp_avg < 1.0) xducer_temp_avg = cur_xducer_temp;
     if (xducer_temp_max < cur_xducer_temp) xducer_temp_max = cur_xducer_temp;
     xducer_temp_avg = ((xducer_temp_avg*9.0)+cur_xducer_temp)/10.0;

     // create avg tank temp
     //avg_tank_temp =
(cur_tank_temp_1+cur_tank_temp_2+cur_tank_temp_3+cur_tank_temp_4)/4.0;    //removed
9/10/02
     avg_tank_temp = cur_tank_temp_1;

     // calc mass
     Hc = HcBaro ( cur_xducer_sensor , cur_local_baro , sg );
     Hc = HcTemp ( tank_dia, initial_temp, avg_tank_temp, sg, Hc, coef_of_expn);
     mass = Hc;
     lastminute = minute;

     // check for room in RAM, write out record in flash if full
     check_local_store();            // reset local data store

     // save results
     index = getcurrentindex();            // get index for data
     data.f.mass[index] = mass;
     data.f.baro[index] = (int)(cur_local_baro*1000); //cur_xducer_sensor*2307); changed
2/18/02
     data.f.avg_temp[index] = (int)(avg_tank_temp*100);
     data.f.xducer_temp[index] = (int)(cur_xducer_temp*100);
    }
    waitfor(DelayMs(250));
   }

  // run heater relay
  costate heater_relay always_on {
   if (cur_pod_temp < desired_xducer_temp-0.1) {
    eioBrdDO(BankB(0),1);     // turn on output
    heater = 1;
   }
   if (cur_pod_temp > desired_xducer_temp+0.1) {
    eioBrdDO(BankB(0),0);     // turn off output
    heater = 0;
```

```
  }
  waitfor (DelayMs(1000));
}

// display heater status by blinking on board LED when heater should be on
costate heater_status always_on {
  if (heater) {
    switchLED(0);
    waitfor (DelayMs(100));
    switchLED(1);
    waitfor (DelayMs(100));
  }else{
    switchLED(1);
    waitfor (DelayMs(100));
    switchLED(0);
    waitfor (DelayMs(2000));
  }
}

// do serial output to baro sensor
costate baro_ouput always_on {
        // if output string is non-zero in length, output it to the sensor
        if (strlen(baro_cmnd)>0) {
    //printf("S:%s:\n", baro_cmnd);
          aascFlushRdBuf(chan1);                        // flush out input buffer
    baro_resp[0]=0;                                           // zero out response string
          aascWriteBlk(chan1,baro_cmnd,strlen(baro_cmnd),1);
          aascWriteBlk(chan1,eol,2,1);              // send end of line
          baro_cmnd[0]=0;                                          // zero out command
  }
  yield;
}

// do serial input from baro sensor
costate baro_input always_on {
  if (aascScanTerm(chan1, 0x0a)>0){
    pos = aascReadBlk(chan1, baro_resp,sizeof(baro_resp),0);
    baro_resp[pos] = 0;                                  // mark end of received string
    // go down string, end string at first <cr>
    for (i=0; i<pos; i++) if (baro_resp[i]==13) baro_resp[i]=0;
    //printf("R:%s:\n", baro_resp);
    aascFlushRdBuf(chan1);
  }
  yield;
}
```

```
// do serial output to terminal
costate term_ouput always_on {
        // if output string is non-zero in length, output it to the sensor
        if (strlen(term_out)>0 && (aascWriteBufFree(chan4) > strlen(term_out)+3) ) {
         aascWriteBlk(chan4,term_out,strlen(term_out),1);
   if (term_out[strlen(term_out)-1]!='*')
           aascWriteBlk(chan4,eol,2,1);               // send end of line
         term_out[0]=0;                                                  // zero out command
 }
 yield;
}


// do serial input from terminal
costate term_input always_on {
 if (aascScanTerm(chan4, 0x0d)>0){
   pos = aascReadBlk(chan4, term_in,sizeof(baro_resp),0);
   term_in[pos] = 0;                                   // mark end of received string
   // go down string, end string at first <cr>
   for (i=0; i<pos; i++) if (term_in[i]==13) term_in[i]=0;
   //printf("Rt%d:%s:\n", strlen(term_in), term_in);
   aascFlushRdBuf(chan4);
 }
 yield;
}

// do modbus processing
costate modbus_io always_on{
 msRun();
 yield;
}

// do terminal processing
costate terminal_io always_on {
 if (strlen(term_in)>0) {                    // process command
   for (i=0; i<strlen(term_in); i++) term_in[i]=toupper(term_in[i]);
   if (strcmp(term_in,"CI")==0) {            // current inputs command?
     sprintf(term_out,"\r\nTrans: %e\r\nTrans Temp :
%f",cur_xducer_sensor,cur_xducer_temp);
     waitfor(strlen(term_out)==0);
     sprintf(term_out,"Baro: %f", cur_local_baro);
     waitfor(strlen(term_out)==0);
     sprintf(term_out,"Heatsink Temp: %f", cur_pod_temp);
     waitfor(strlen(term_out)==0);
     sprintf(term_out,"Ambient Temp: %f\r\nProduct 1 Temp:
```

```
%f",cur_tank_temp_1,cur_tank_temp_2);
      waitfor(strlen(term_out)==0);
      sprintf(term_out,"Product 2 Temp: %f\r\nProduct 3 Temp:
%f",cur_tank_temp_3,cur_tank_temp_4);
      waitfor(strlen(term_out)==0);
    }
    if (strcmp(term_in,"CV")==0) {    // current values
      sprintf(term_out,"\r\nCor. Head: %f",mass);
      waitfor(strlen(term_out)==0);
      sprintf(term_out,"Baro: %f", cur_local_baro);
      waitfor(strlen(term_out)==0);
      sprintf(term_out,"Ambient Temp: %f",avg_tank_temp);
      //sprintf(term_out,"\r\nCor. Head: %f\r\nBaro: %f\r\nAmbient Temp: %f", mass,
cur_local_baro, avg_tank_temp);
      waitfor(strlen(term_out)==0);
    }
    if (strcmp(term_in,"XT")==0) {    // xducer temp info
      sprintf(term_out, "\r\nToday");
      waitfor(strlen(term_out)==0);
      sprintf(term_out,"Min: %f\r\nMax: %f\r\nAvg:
%f",xducer_temp_min,xducer_temp_max,xducer_temp_avg);
      waitfor(strlen(term_out)==0);
      sprintf(term_out, "\r\nYesterday");
      waitfor(strlen(term_out)==0);
      sprintf(term_out,"Min: %f\r\nMax: %f\r\nAvg:
%f",xducer_ytemp_min,xducer_ytemp_max,xducer_ytemp_avg);
      waitfor(strlen(term_out)==0);
    }
    if (strncmp(term_in,"DT",2)==0) {    // current date?
      if (term_in[2]=='=') {                              // set date/time
        year =  (term_in[3]-48)*10+term_in[4]-48+2000;
        month = (term_in[5]-48)*10+term_in[6]-48;
        day =   (term_in[7]-48)*10+term_in[8]-48;
        hour =  (term_in[9]-48)*10+term_in[10]-48;
        minute =(term_in[11]-48)*10+term_in[12]-48;
        write_datetime();
      }
      read_datetime();
      sprintf(term_out,"\r\nYear: %d",year);
      waitfor(strlen(term_out)==0);
      sprintf(term_out,"Month: %d\r\nDay:  %d\r\nHour:  %d",month,day,hour);
      waitfor(strlen(term_out)==0);
      sprintf(term_out,"Minute:%d", minute);
      waitfor(strlen(term_out)==0);
    }
```

```
if (strncmp(term_in,"TT",2)==0) {     // transducer temp
  if (term_in[2]=='=') {                                  //  set transducer temp
    u.f = atof(&(term_in[3]));
    i=WriteFlash(phy_adr(&desired_xducer_temp),u.c,sizeof(u.c));
  }
  sprintf(term_out,"\r\nHeatsink Temp:  %f",cur_pod_temp);
  waitfor(strlen(term_out)==0);
  sprintf(term_out,"Setpoint:  %f",desired_xducer_temp);
  waitfor(strlen(term_out)==0);
  sprintf(term_out,"Heater: %s",heater?"ON":"OFF");
  waitfor(strlen(term_out)==0);
}
if (strncmp(term_in,"SD",2)==0) {     // transducer temp
  if (term_in[2]=='=') {                                  //  set transducer temp
    u.f = atof(&(term_in[3]));
    i=WriteFlash(phy_adr(&day_start),u.c,sizeof(u.c));
    start_hour = (int)(day_start/100);
    start_min = (int)(day_start-start_hour*100);
  }
  sprintf(term_out,"\r\nStart of Day: %2d:%2d",start_hour,start_min);
  waitfor(strlen(term_out)==0);
}
if (strncmp(term_in,"MA",2)==0) {     // t
  if (term_in[2]=='=') {
    modbusaddr = atoi(&(term_in[3]));
  }
  sprintf(term_out,"\r\nModbus Addr:  %d",modbusaddr);
  waitfor(strlen(term_out)==0);
}
if (strncmp(term_in,"TD",2)==0) {     // tank dia
  if (term_in[2]=='=') {
    u.f = atof(&(term_in[3]));
    i=WriteFlash(phy_adr(&tank_dia),u.c,sizeof(u.c));
  }
  sprintf(term_out,"\r\nTank Dia:  %f",tank_dia);
  waitfor(strlen(term_out)==0);
}
if (strncmp(term_in,"SG",2)==0) {     // specific gravity
  if (term_in[2]=='=') {
    u.f = atof(&(term_in[3]));
    i=WriteFlash(phy_adr(&sg),u.c,sizeof(u.c));
  }
  sprintf(term_out,"\r\nSpecific G:  %f",sg);
  waitfor(strlen(term_out)==0);
}
```

```
if (strncmp(term_in,"BC",2)==0) {    // baro correction factor
  if (term_in[2]=='=') {
    u.f = atof(&(term_in[3]));
    i=WriteFlash(phy_adr(&Bcf),u.c,sizeof(u.c));
  }
  sprintf(term_out,"\r\nBaro Correct:  %f",Bcf);
  waitfor(strlen(term_out)==0);
}
if (strncmp(term_in,"CE",2)==0) {    // coef of expansion
  if (term_in[2]=='=') {
    u.f = atof(&(term_in[3]));
    i=WriteFlash(phy_adr(&coef_of_expn),u.c,sizeof(u.c));
  }
  sprintf(term_out,"\r\nCoef of Expn:  %f",coef_of_expn);
  waitfor(strlen(term_out)==0);
}
if (strncmp(term_in,"IT",2)==0) {    // initial temp
  if (term_in[2]=='=') {
    u.f = atof(&(term_in[3]));
    i=WriteFlash(phy_adr(&initial_temp),u.c,sizeof(u.c));
  }
  sprintf(term_out,"\r\nInitial Temp: %f",initial_temp);
  waitfor(strlen(term_out)==0);
}
if (strncmp(term_in,"CS",2)==0) {    // clear storage
  for (rr=0; rr<36; rr++) {
    data.f.year = 0;
    data.f.day = 0;
    data.f.hour = 0;
    data.f.initial1 = 1234;
    data.f.initial2 = rr;
    flash_write(rr);
  }
  sprintf(term_out,"\r\nAll data storage has been cleared.");
  waitfor(strlen(term_out)==0);
}
if (strncmp(term_in,"DD",2)==0) {    // dump data in comma delimited format
  strcpy(term_out, "\r\n");
  waitfor(strlen(term_out)==0);
  // start with ram record, and go back up until we've reached zero
  for (ii=index; ii>=0; ii--) {
    if (data.f.baro[ii]!=0 || data.f.avg_temp[ii]!=0) { // print this record
      if (formatdt(-1,ii)>0) {
        sprintf(term_out,"%s, %f, %d, %d, %d",date_time,data.f.mass[ii],
                  data.f.baro[ii],data.f.avg_temp[ii],data.f.xducer_temp[ii]);
```

```
          waitfor(strlen(term_out)==0);
        }
      }
    }
    // pickup the previous record and print it also,
    // repeat until we've come back to the current record.
    rr = current_record - 1;
    do {
      yield;
      flash_read(rr);
      for (ii=240-1; ii>=0; ii--) {
        if ((old.f.baro[ii]!=0) || (old.f.avg_temp[ii]!=0)) { // print this record
          if (formatdt(rr,ii)>0){
            sprintf(term_out,"%s, %f, %d, %d, %d",date_time,old.f.mass[ii],
                    old.f.baro[ii],old.f.avg_temp[ii], old.f.xducer_temp[ii]);
            waitfor(strlen(term_out)==0);
          }
        }
      }
    rr--;
    if (rr<0) rr=36-1;
    } while (rr != current_record);
   }
   sprintf(term_out,"*");
   waitfor(strlen(term_out)==0);
   term_in[0]=0;
  }
 yield;
 }

 // run serial baro sensor
 costate baro_io always_on {
  strcpy(baro_cmnd,"*0100P3");
  waitfor(DelaySec(45));
  if (strlen(baro_resp)>6) strcpy(serial_pressure,&(baro_resp[5]));
  else strcpy(serial_pressure,"0.0");
  cur_xducer_sensor = atof(serial_pressure);
  strcpy(baro_cmnd,"*0100Q3");
  waitfor(DelaySec(10));
  if (strlen(baro_resp)>6) strcpy(serial_temp,&(baro_resp[5]));
  else strcpy(serial_temp,"0.0");
  cur_xducer_temp = atof(serial_temp)*9.0/5.0+32.0;
  waitfor(DelaySec(4));
 }
}
```

```
}

// get *E*ven word for *F*loating point modbus register
unsigned int EF(float f){
 modbusd.f = f;
 return modbusd.i[0];
}

// *O*dd *W*ord
unsigned int OF(float f){
 modbusd.f = f;
 return modbusd.i[1];
}

// write *E*ven word of modbus bus floating point write
void EW(float *f, int i){
 modbusd.i[0] = i;
 *f = modbusd.f;
}

void OW(float *f, int i){
 modbusd.i[1] = i;
 *f = modbusd.f;
}

/*=================================================================
=====*\
      Read Output Coil (Registers 00001-00020)
\*=================================================================
=====*/

int
msOutRd ( unsigned   wCoil,
     int      *pnState
    )

{
 return MS_BADADDR;
}

/*=================================================================
=====*\
      Write Output Coil (Registers 00001-00020)
\*=================================================================
=====*/
```

```
int
msOutWr ( unsigned    wCoil,
          int         bState
        )

{
 return MS_BADADDR;
}

/*=================================================================
=====*\
        Read Input Coil (Registers 10001-10020)
\*=================================================================
=====*/

int
msIn ( unsigned   wCoil,
       int        *pnState
     )

{
 return MS_BADADDR;
}

/*=================================================================
=====*\
        Read Input Register (Registers 30001-3000A)
\*=================================================================
=====*/

int
msInput ( unsigned    wReg,
          unsigned    *pwValue
        )

{
 return MS_BADADDR;
}
/*=================================================================
=====*\
     Read Register 40000
\*=================================================================
=====*/

/*** BeginHeader msRead */
```

```
int msRead ( unsigned wReg,unsigned *pwValue );
/*** EndHeader */

int
msRead ( unsigned  wReg,
      unsigned  *pwValue
     )
{
 int rvalue;
 int recnum, item;
 int currec;

 rvalue = 0;
 switch (wReg) {
  case 0:
   *pwValue = year;
   break;
  case 1:
   *pwValue = month;
   break;
  case 2:
   *pwValue = day;
   break;
  case 3:
   *pwValue = hour;
   break;
  case 4:
   *pwValue = minute;
   break;
  case 10:
   *pwValue = EF(desired_xducer_temp);
   break;
  case 11:
   *pwValue = OF(desired_xducer_temp);
   break;
  case 12:
   *pwValue = EF(tank_dia);
   break;
  case 13:
   *pwValue = OF(tank_dia);
   break;
  case 14:
   *pwValue = EF(sg);
   break;
  case 15:
```

```
  *pwValue = OF(sg);
  break;
 case 16:
  *pwValue = EF(coef_of_expn);
  break;
 case 17:
  *pwValue = OF(coef_of_expn);
  break;
 case 18:
  *pwValue = EF(initial_temp);
  break;
 case 19:
  *pwValue = OF(initial_temp);
  break;
 case 20:
  *pwValue = start_hour;
  break;
 case 21:
  *pwValue = start_min;
  break;
 case 22:
  *pwValue = EF(Bcf);
  break;
 case 23:
  *pwValue = OF(Bcf);
  break;
 case 100:
  *pwValue = EF(mass);  // ??
  break;
 case 101:
  *pwValue = OF(mass);  // ??
  break;
 case 102:
  *pwValue = EF(cur_xducer_temp);
  break;
 case 103:
  *pwValue = OF(cur_xducer_temp);
  break;
 case 104:
  *pwValue = EF(cur_tank_temp_1);
  break;
 case 105:
  *pwValue = OF(cur_tank_temp_1);
  break;
 case 106:
```

```
    *pwValue = EF(cur_tank_temp_2);
    break;
  case 107:
    *pwValue = OF(cur_tank_temp_2);
    break;
  case 108:
    *pwValue = EF(cur_tank_temp_3);
    break;
  case 109:
    *pwValue = OF(cur_tank_temp_3);
    break;
  case 110:
    *pwValue = EF(cur_tank_temp_4);
    break;
  case 111:
    *pwValue = OF(cur_tank_temp_4);
    break;
  case 112:
    *pwValue = EF(xducer_temp_min);
    break;
  case 113:
    *pwValue = OF(xducer_temp_min);
    break;
  case 114:
    *pwValue = EF(xducer_temp_max);
    break;
  case 115:
    *pwValue = OF(xducer_temp_max);
    break;
  case 116:
    *pwValue = EF(xducer_temp_avg);
    break;
  case 117:
    *pwValue = OF(xducer_temp_avg);
    break;
  case 118:
    *pwValue = EF(xducer_ytemp_min);
    break;
  case 119:
    *pwValue = OF(xducer_ytemp_min);
    break;
  case 120:
    *pwValue = EF(xducer_ytemp_max);
    break;
  case 121:
```

```
    *pwValue = OF(xducer_ytemp_max);
    break;
  case 122:
    *pwValue = EF(xducer_ytemp_avg);
    break;
  case 123:
    *pwValue = OF(xducer_ytemp_avg);
    break;
  case 124:
    *pwValue = EF(cur_local_baro);
    break;
  case 125:
    *pwValue = OF(cur_local_baro);
    break;
  case 126:
    *pwValue = EF(cur_xducer_sensor);
    break;
  case 127:
    *pwValue = OF(cur_xducer_sensor);
    break;
  default:
    rvalue = MS_BADADDR;
    break;
}

if (wReg >= 1000) {

  recnum = wReg/1000;
  item = wReg%1000;
  currec = getcurrentindex();
  if (recnum < currec) {   // use current ram data
    switch (item) {
     case 0:
       *pwValue = data.f.year;
        break;
     case 1:
       *pwValue = data.f.month;
       break;
     case 2:
       *pwValue = data.f.day;
       break;
     case 3:
       if (data.f.hour<12)
         *pwValue = (int)(recnum*2)/60;
       else
```

```
      *pwValue = 12 + (int)(recnum*2)/60;
    break;
   case 4:
    *pwValue = (recnum * 2)%60;
    break;
   case 6:
    *pwValue = EF(data.f.mass[currec-recnum-1]);
    break;
   case 7:
    *pwValue = OF(data.f.mass[currec-recnum-1]);
    break;
   case 8:
    *pwValue = data.f.baro[currec-recnum-1];
    break;
   case 9:
    *pwValue = data.f.avg_temp[currec-recnum-1];
    break;
  }
  rvalue = 0;
 }
 else {                                        // use flash data
  flash_read(getcurrentindex());
  switch (item) {
   case 0:
    *pwValue = old.f.year;
     break;
   case 1:
    *pwValue = old.f.month;
    break;
   case 2:
    *pwValue = old.f.day;
    break;
   case 3:
    if (old.f.hour<12)
      *pwValue = (int)((currec-recnum-1)*2)/60;
    else
      *pwValue = 12 + (int)((currec-recnum-1)*2)/60;
    break;
   case 4:
    *pwValue = ((currec-recnum-1) * 2)%60;
    break;
   case 6:
    *pwValue = EF(old.f.mass[currec-recnum-1]);
    break;
   case 7:
```

```
      *pwValue = OF(old.f.mass[currec-recnum-1]);
      break;
     case 8:
      *pwValue = old.f.baro[currec-recnum-1];
      break;
     case 9:
      *pwValue = old.f.avg_temp[currec-recnum-1];
      break;
    }
    rvalue = 0;
  }
 }

// return rvalue;
 return 0;
}

/*=============================================================
=====*\
        Write Register
\*=============================================================
=====*/

/*** BeginHeader msWrite */
int msWrite ( unsigned wReg,unsigned wValue );
/*** EndHeader */

int
msWrite ( unsigned  wReg,
       unsigned  wValue
      )
{
 int rvalue, i;
 union { float f; char c[4]; } u;

 rvalue = 0;
 switch (wReg) {
  case 0:
   year = wValue;
   write_datetime();
   break;
  case 1:
   month = wValue;
   write_datetime();
   break;
```

```
case 2:
 day = wValue;
 write_datetime();
 break;
case 3:
 hour = wValue;
 write_datetime();
 break;
case 4:
 minute = wValue;
 write_datetime();
 break;
case 10:
 u.f = desired_xducer_temp;
 EW(&u.f, wValue);
 i=WriteFlash(phy_adr(&desired_xducer_temp),u.c,sizeof(u.c));
 break;
case 11:
 u.f = desired_xducer_temp;
 OW(&u.f, wValue);
 i=WriteFlash(phy_adr(&desired_xducer_temp),u.c,sizeof(u.c));
 break;
case 12:
 u.f = tank_dia;
 EW(&u.f, wValue);
 i=WriteFlash(phy_adr(&tank_dia),u.c,sizeof(u.c));
 break;
case 13:
 u.f = tank_dia;
 OW(&u.f, wValue);
 i=WriteFlash(phy_adr(&tank_dia),u.c,sizeof(u.c));
 break;
case 14:
 u.f = sg;
 EW(&u.f, wValue);
 i=WriteFlash(phy_adr(&sg),u.c,sizeof(u.c));
 break;
case 15:
 u.f = sg;
 OW(&u.f, wValue);
 i=WriteFlash(phy_adr(&sg),u.c,sizeof(u.c));
 break;
case 16:
 u.f = coef_of_expn;
 EW(&u.f, wValue);
```

```
    i=WriteFlash(phy_adr(&coef_of_expn),u.c,sizeof(u.c));
    break;
  case 17:
    u.f = coef_of_expn;
    OW(&u.f, wValue);
    i=WriteFlash(phy_adr(&coef_of_expn),u.c,sizeof(u.c));
    break;
  case 18:
    u.f = initial_temp;
    EW(&u.f, wValue);
    i=WriteFlash(phy_adr(&initial_temp),u.c,sizeof(u.c));
    break;
  case 19:
    u.f = initial_temp;
    EW(&u.f, wValue);
    i=WriteFlash(phy_adr(&initial_temp),u.c,sizeof(u.c));
    break;
  case 20:
    start_hour = wValue;
    i = day_start - (int)(day_start / 100)*100;
    u.f = start_hour + i;
    i=WriteFlash(phy_adr(&day_start),u.c,sizeof(u.c));
    break;
  case 21:
    start_min = wValue;
    u.f = ((int)(day_start/100))*100 + start_min;
    i=WriteFlash(phy_adr(&day_start),u.c,sizeof(u.c));
    break;
  case 22:
    u.f = Bcf;
    EW(&u.f, wValue);
    i=WriteFlash(phy_adr(&Bcf),u.c,sizeof(u.c));
    break;
  case 23:
    u.f = Bcf;
    OW(&u.f, wValue);
    i=WriteFlash(phy_adr(&Bcf),u.c,sizeof(u.c));
    break;
  default:
            rvalue = MS_BADADDR;
    break;
 }
 return rvalue;
}
```